# A Path Planning Method Based on Genetic Algorithms and Curve Fitting Techniques

Brock Kopp
Mechatronics Engineering
University of Waterloo
Waterloo, Ontario
brock.kopp@uwaterloo.ca

Karl Price
Mechatronics Engineering
University of Waterloo
Waterloo, Ontario
karldprice@gmail.com

Angelica Ruszkowski
Mechatronics Engineering
University of Waterloo
Waterloo, Ontario
aruszkow@uwaterloo.ca

***Abstract*—Navigation and path planning are critical components of developing robotic technology, where optimized path planning can improve efficiency and throughput of an industrial robotic manipulator. A genetic algorithm is used to plan a polynomial function based trajectory which avoids collisions with any obstacles and minimizes path distance and dynamic constraints. The stochastic nature of the genetic algorithm allows it to determine a viable and efficient path for a manipulator quickly as its environment changes. Algorithm results are verified by comparing with deterministic algorithms which produce the absolute best path.**

***Index Terms*—Genertic Algorithms, Curve Fitting, Path Planning, Grid Method, Robotics**

## I. Introduction

Path planning is a critical component of navigation technology, and a fundamental area in robotics research. Verifying path safety is critical for both the welfare of the robot as well as its surrounding environment, which can contain people or valuable infrastructure. The path must ultimately reach its destination while satisfying specified criteria, the most common of which is distance (or time). Another important criterion is path dynamics, where the forces and other dynamic characteristics of the robot are considered. Excessive acceleration and jerk cause stress on the robotic manipulator and degrade robot tracking characteristics, which are often a function of acceleration. These criteria are accounted for by relatively few common path planning algorithms [**?**].

Path planning techniques can be applied to industrial robotic manipulators to ensure efficient operation while maneuvering safely in the environment. Articulated manipulators are bounded to their configuration workspace, which is a two-dimensional representation of their allowable motion range [**?**]. Frequently used algorithms for motion planning include probabilistic road maps, potential field methods, and neural network approaches [**?**], [**?**], [**?**], [**?**]. Since every robot has a unique and complex task and environment, a robust solution is necessary. Genetic algorithms are especially suited for such complex optimization problems [**?**].

As a stochastic optimization technique, it is expected that a genetic algorithm will find the global optimum for a problem in a reasonable amount of time and computational cost. Due to the iterative evolutionary nature of genetic algorithms however, computational time may still be greater than other probabilistic algorithms. Overarching these concerns, the robustness of the genetic algorithm, combined with its ability to easily evaluate all path criteria of the optimization problem make it an appealing solution.

The robot's environment and obstacles will be represented in a discretized grid, and polynomial curve fitting techniques will be used to define the optimum path from the start to the end location through the specified environment. The curve fitting method helps simplify and expedite the generation of a desired path, compared to commonly used techniques such as rapidly exploring random trees [**?**]. The genetic algorithm will determine the polynomial coefficients which result in a curve with the greatest fitness, as determined by the constraints of the problem (namely path distance and jerk). A polynomial function was selected to describe the path since it inherently results in a continuous trajectory rather than traditional discontinuous way point based trajectories.

While there is no absolute best path planning algorithm for all situations, it is proposed that:

- A Genetic Algorithm can effectively plan a path through a robotic manipulator's configuration space, producing a valid result 94% of the time.
- The algorithm presented can plan a path which not only minimizes distance traveled, but also accounts for robotic manipulator dynamics such as acceleration and jerk.
- The algorithm can identify a viable solution which is within 1% of the shortest distance as determined using a deterministic algorithm such as wavefront, while showing improvements to computation time.

## II. Background Review

Path planning is an active area of research due to the complexity of the problem and the need for a robust solution applicable to various environments. Many algorithms exist, but none of the proposed algorithms are capable of encompassing all the problem constraints for all environments [**?**].

Existing path planning solutions include reactive motion planning algorithms, graph and probabilistic based motion planning, and optimization based planning [**?**]. Each of these has advantages and disadvantages with regards to planning a route through a robot's environment. Reactive planning algorithms such as potential field and wavefront methods are simple in concept however these are local approach methods and do not consider dynamic constraints. Some are inherently susceptible to local minima, making them unacceptable for obstacle avoidance [?].

The wavefront method is designed to find the absolute shortest path, and as such will be used as the benchmark against which results from the suggested algorithm will be measured. It is expected that while the suggested algorithm will generate a longer path, the dynamic constraints will be satisfied, unlike the results of the wavefront solution.

Graph based planning offers a global approach. Graphs may be generated both deterministically (visibility graphs, cell decomposition, Voronoi diagrams), or randomly (probabilistic road maps). Established algorithms (such as Dijkstra's shortest path search) are used to extract the shortest path from the graph [?]. However dynamics of the robot are ignored in the generation of the graph, and this is detrimental to the criterion of path smoothness. Challenges also arise when there are restrictive obstacles such as narrow passages, however research in areas such as probabilistic road maps (PRMs) offers promising solutions [?]. Optimization based planning offers more confidence in finding a global optimum, however it is hindered by a lack of robustness: a poorly formulated problem may never converge [?]. It is also difficult to define obstacles.

In an effort to provide a simple method of avoiding obstacles and defining a path, the manipulator's configuration space will be discretized using a grid-based approach, and curve fitting techniques will be employed to find the ideal path. The configuration space of a robotic manipulator is a transformation of the manipulator's joint space to a two-dimensional Cartesian space. Defining the environment with a grid-graph may not be the most effective (cell decomposition offers greater efficiency [?]), however it provides a simple way to define obstacles as well as flexibility in representation. The grid will represent the configuration space for an articulated industrial robot. The grid cell size will define the computational resources required. A trade-off between resolution in the grid and computational efficiency must be made.

Bézier curves have been suggested to define the fitted curves generated by the genetic algorithm [?]. Bézier curves provide smooth paths between two vectors, and while very effective for lower-dimensional problems, Bézier functions become very computationally expensive as the problem dimensionality increases. This paper therefore aims to use polynomial curve fitting in combination with genetic algorithms to optimize the curve. The curve will be optimized by minimizing distance, maximizing smoothness (minimizing jerk), and ensuring a safe passage around all obstacles. Genetic algorithms present an appealing approach for such complex optimization problems and offer good robustness as well [?].

The father of genetic algorithms, John Holland, modeled this machine intelligence technique after Darwin's Theory of Evolution, and the concept of "survival of the fittest" which he observed in nature. There are two natural learning epitomes available: the brain and evolution. Holland was inspired by the fact that the processes of natural evolution and natural genetics have become elucidated over decades of study in biology and molecular biology [?]. The subtleties of the fundamental mechanisms of the brain, in contrast, are still shrouded in mystery. As such it seems clear to use the better understood model of genetic evolution as a platform for an optimization technique.

Research has already been done regarding the application of the parallel heuristic search method of genetic algorithms to curve fitting [?], [?], [?]. This paper builds upon that research. The goal is to find the coefficients of a polynomial function that will define a path with optimized path criteria. Thus each chromosome will contain a set of function parameters, labeled $\{P_3, \ldots, P_0\}$ in Equation 1.

$$\phi = P_3 * \theta^3 + P_2 * \theta^2 + P_1 * \theta + P_0 \tag{1}$$

Testing will be performed by creating random environments containing arbitrarily placed objects. The algorithm will then plot a trajectory between a randomly defined start and end point. Many iterations of these tests will be performed in a variety of environments in order to accurately determine the effectiveness of this algorithm. The average path distance for each environment will be compared to the minimum distance as calculated by a Wavefront algorithm [?].

The wavefront algorithm is a deterministic algorithm which is capable of finding the shortest path between two points through a discretized environment. This will provide a "gold standard" which can be used to evaluate the performance of the genetic algorithm. The wavefront implementation used in this study permits only 90 or 45 degree turns through the environment. A sample trajectory defined by a wavefront algorithm is presented in Figure 1. Since the genetic algorithm is a stochastic search, it will not result in an identical path twice. As such all results will be presented as averages and their associated distributions.

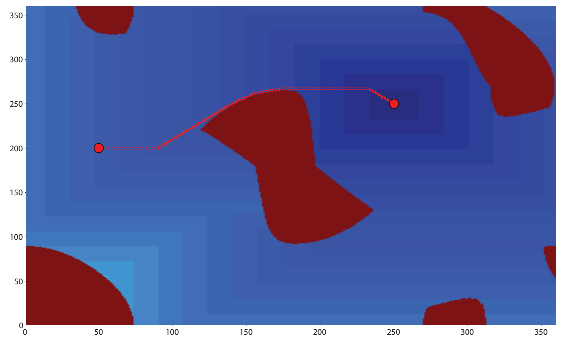

Fig. 1. The path as determined by the wavefront algorithm used to evaluate the performance of the genetic algorithm. The origin of the path is denoted by the darker blue, and the "wave" propogation is shown by increasingly light blue coloured cells.

The result will be a global path planning algorithm functional in a static environment. Dynamic environments are possible with the proposed algorithm, however computational efficiency must be optimized in order to minimize the time required to find the solution. This new application offers a simple way to determine a path while taking into consideration all the defined path criteria.

## III. Method

The Genetic Algorithm (GA) was the subject of the experiments. Significant effort was expended setting up the GA in advance of testing, since without a well-structured algorithm, the testing was sure to show poor results. The GA was implemented in MATLAB, using the built-in GA libraries in order to avoid unnecessary development. While the base functionality was used, there remained many design considerations.

### A. Criteria and Fitness

The aim of the implementation was to produce a path which minimized obstacle collisions, length and jerk of the path represented by a polynomial function. The fitness of an individual was based on an equation considering these three criteria.

In order to detect obstacle collisions, the algorithm iterated along the path, and checked at each point to see if the path was inside an obstacle. The chromosome was penalized proportionally to the number of path units within an obstacle. Collisions resulted in high punishments to the fitness value of an individual. It was important to divert the path around a collision point, as a single collision in the final solution would render that entire path invalid.

Length was calculated based on the integral line length formula. The weight of the line length was set to a medium penalty (relative to collisions and jerk). This was done since it is more important to find a valid solution that does not collide with any obstacles, than find the shortest path between the two points.

Finally, the maximum jerk value of the line was found by iterating the length of the fourth derivative of the path equation and recording the maximum value of this derivative. The jerk weight was set to make reduction in the jerk as important as reduction in the path length.

### B. Configuration Space

Commanding a robotic manipulator's position is complicated by the fact that it cannot be specified as X-Y-Z coordinates, but rather must be commanded using joint angles. This is not intuitive for a human, and many existing planning techniques function using the more human-oriented X-Y or X-Y-Z coordinate systems, forcing the human to then convert their solutions to a joint space solution. As such, the robot's workspace was converted into a Configuration Space to allow the genetic algorithm to work with joint angles. Figure 2 shows both the workspace and the configuration space for a 2-DOF robot in an arbitrary workspace. The robotic manipulator's two degrees of freedom are denoted by the blue and red lines (each DOF respectively). The allowable robot configurations are represented in the Configuration Space by white areas, and collisions are represented by black.

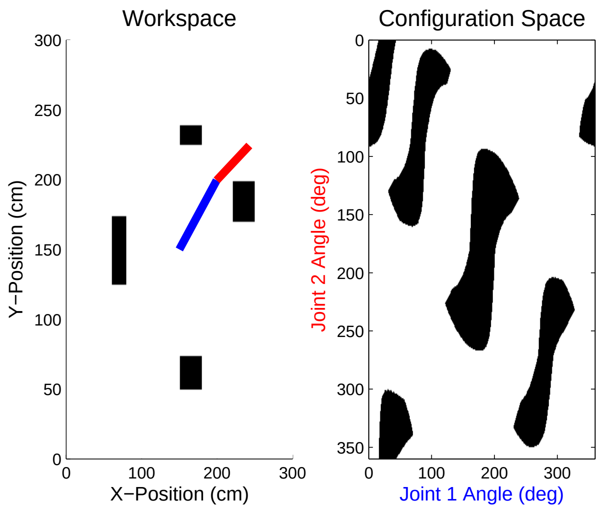

Fig. 2. Conversion from robot workspace to configuration space

In the configuration space, robot trajectories can be easily identified by either a human or a motion planning algorithm. This angle-angle plot can then be used to implement a polynomial trajectory, of which optimal coefficients can easily be solved using a Genetic Algorithm. The parameterized trajectory is denoted in Equation 1.

### C. Population Initialization

The population was randomly initialized with coefficient values in the range of -500 to 500. This range was imposed on the coefficients as it was found to provide a good compromise between promoting function variation while at the same time keeping it mitigated enough so that changes to coefficients each generation did not make too drastic a change to the shape of the path.

The population size was chosen to be 75, after research found that exiting solutions consistently used a population range of approximately 50-100 [?]. This population number provides the necessary diversity in the population. Lower population values were found to converge to non-ideal solutions.

### D. Encoding

The GA was developed using a linear encoding scheme, where a chromosome was defined to be an array of the polynomial function's coefficients (denoted as $\{P_0, \ldots, P_3\}$ in Equation 1). Therefore, the GA would ultimately return the ideal set of coefficients for a function which would describe the optimal path for the robotic manipulator.

### E. Linear Constraints

The path planning problem is a challenging optimization problem due to the various constraints involved. The goal is to generate a path that avoids all obstacles, as well as minimize the path length and minimize jerk. The problem is further constrained by the start and end points. The generated polynomial path must go through these desired start and end points.

To solve this constrained optimization problem, linear equality constraints are used. Linear equalities have the form:

$$A_{eq}x = b_{eq} \tag{2}$$

Where x is a vector of the variables being optimized. For this project, the optimization variables are the coefficients of the polynomial.

For example, a second order polynomial $y = A+Bx+Cx^2$ through points $[2, 4]$ and $[10, 5]$ would have linear constraints of the form:

$$\begin{bmatrix} 1 & 10 & 10^2 \\ 1 & 2 & 2^2 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \end{bmatrix} \tag{3}$$

### F. Selection

A tournament selection method was used in this implementation to select chromosomes for cross-over. Two individuals were chosen for each tournament, allowing for an acceptable tradeoff between choosing a fit parent while promoting diversity in the offspring. This was one approach used to increase the diversity of the population when it was observed that the algorithm was getting stuck in invalid local minima.

### G. Reproduction and Mutation

A heuristic reproduction method was used wherein the child is generated by taking the mean of the coefficients of the two parents, and then biased to resemble one of its parents. In this implementation, the child was biased 20% towards the most fit parent using Formula 4.

$$child_i = parent_{i2} + 1.2 * (parent_{i1} - parent_{i2}) \tag{4}$$

As is common in non-binary encoded GA problems, the crossover method used in this implementation is based on a mean of the parents' genes. This is a key distinction from a traditional point crossover method, as it allows the genes to evolve from values that were not present in the initial population. If point crossover methods had been used, the best case scenario would simply involve the optimal configuration of coefficients from the initial, random population.

Mutation was achieved applying a random mutation to a coefficient within the individual selected to be mutated. In order to satisfy the linear constraints of the start and end points, if the function was rendered invalid by the mutation, a new random coefficient was chosen instead. The mutated coefficient must lie within the defined coefficient range of -500 to 500.

### H. Termination

The termination criteria allow the GA to determine when it has reached the optimal solution to the problem. In this implementation, a termination criteria based on convergence of the fitness function was used. The mean fitness value over ten generations was averaged. When the averaged mean fitness value converged within 0.001 units, the algorithm was terminated. This is representative of a situation where the diversity of the function has been reduced, and the algorithm has converged on a minimum fitness value.

### I. Experiment

In order to assess the performance of the GA approach, an extensive experiment was devised. The algorithm was run 600 times on 15 sets of randomly generated points in 3 environments (forming 45 different environments), with randomly positioned obstacles. Invalid combinations of point sets (such as a point which lies within an obstacle) were removed from the dataset and new points were randomly generated to replace them. The environments were chosen to represent environments which were sparsely populated with obstacles, through to dense population with obstacles. The results obtained by this experiment are presented Section IV.

## IV. Results

The following results were obtained through an experiment of 600 independent trials of the genetic algorithm in different environments. Robot configuration spaces were generated based on physical environments which featured objects randomly placed in the robot's path.

The number of test environment configurations was limited not by the genetically algorithm, but rather by the wavefront algorithm being used to validate results. Since the wavefront algorithm runtime approached 20 minutes per configuration, it was prohibitive to evaluate large numbers of possible environments in the available time.

Three configuration spaces were identified as good candidates for testing purposes, based on maps of varying complexity and obstacle density. Within each of these three environments, five sets of path start and end points were generated randomly. The points were checked to ensure that no invalid point configurations were present (i.e. conflicts situated within an obstacle). The algorithm was then run 40 times on each point set, and with five point sets per environment and three environments, a total of 600 trials were conducted.

Figure 3 shows a visualization of 200 trials conducted on five start-end point sets on a populated configuration space. Note that each line in the visualization represents a full evolution cycle with a population of 75 individuals. While individual paths are obscured, the line densities show that the algorithm was able to determine a short and smooth trajectory in the majority of cases, with few outliers. The selected random point configurations exhibit both obscured and trivial problems (representative of a real-world environment). The results of the experiment are presented in Table I.

## V. Discussion

Evaluating the performance of the genetic algorithm (GA) was complicated by its ability to consider many factors when

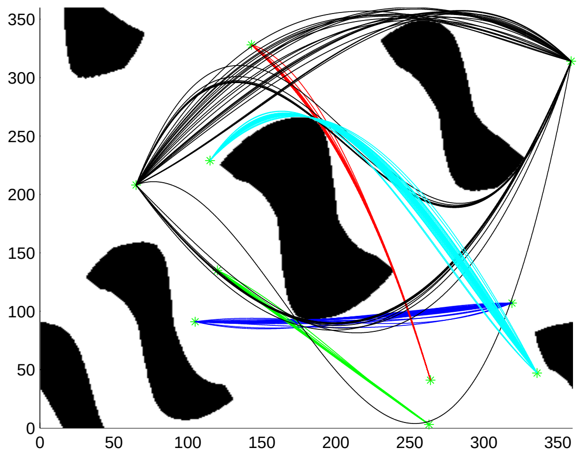

Fig. 3. Resulting paths from 200 independent trials, between five start-end point configurations in a single environment. Each path is the result of a full life-cycle of the algorithm. Paths show the algorithm consistently identifies the shortest path, with few outliers.

TABLE I
AGREGATED RESULTS OF 600 TRIALS OF THE PATH FINDING ALGORITHM, BETWEEN 15 ENVIRONMENT CONFIGURATIONS.

| Criteria | Result | Unit | Note |
|---|---|---|---|
| Path length | 0.7 | % longer | Compared to wavefront method |
| Freq. of shorter path | 79 | % | Compared to wavefront method |
| Time to completion | 5.15 | seconds | 20767% improvement on wavefront method |
| Average number of generations | 12.3 | generations | |
| Freq. of collision-free path | 94 | % | |
| Jerk improvement | 194 | $\frac{D}{T^3}$ | Compared to implementation not minimizing jerk. |

determining the best path. These factors included path distance, vehicle dynamics and most importantly, obstacle avoidance. In order to determine a profile of the GA's performance, 600 independent tests were completed within 15 different environment configurations.

The length of the path generated by the GA approach was on average 0.7% longer than the path generated by the wavefront method to which it was compared. The GA was able to determine a shorter path than the wavefront algorithm in 474 of 600 independent tests (79%), while also minimizing manipulator jerk (sometimes at the cost of a longer path). The wavefront is a deterministic algorithm which will find a very short path around obstacles, with no consideration for vehicle dynamics. The GA works differently in that it is a stochastic algorithm that is sampling the environment in an effort to find the shortest path. As such, it can get stuck in a local minimum and not ultimately return the path of best possible fitness. While processes internal to the GA attempt to avoid these situations, it is not a perfect algorithm.

The GA was found to be significantly faster at finding a solution than the wavefront algorithm. The wavefront algorithm took an average of 17 minutes and 50 seconds to analyze the shortest path between two points in any of the three test spaces used in this study. In comparison, the genetic algorithm took an average of 5.15 seconds to find the optimal path, or roughly a 20700% improvement.

While the absolute execution time of the algorithm (in seconds) is arbitrary and based on algorithm implementation, an improvement in execution time exceeding two orders of magnitude show a very conclusive improvement in performance over the wavefront algorithm. Both of these tests were performed using Matlab on a consumer grade desktop computer. With execution times in the seconds, it is possible that this algorithm could be implemented in real-time on a robotic manipulator whose environment was liable to change infrequently.

The algorithm was found to converge to a solution within an average of 12.3 generations, with a standard deviation of 2.58 generations. This represents a fast convergence of the algorithm to a single solution.

It was expected to be able to develop an algorithm that would find a valid path which never collided with an obstacle. The results from the experiment show that the algorithm found a collision-free path 565 times in 600 (94%) of the time. While this was not the anticipated result, it still constitutes a successful solution to the problem. In an industry application, if the algorithm produces a path which collides with an obstacle, it would simply re-initialize the population and start the evolution process again. With a run time of 5.15 seconds, restarting the algorithm would not be prohibitive for most applications.

Since trajectories varied significantly between environments and start/end locations, a reference best-case scenario jerk could not be identified. The jerk of a path was therefore compared to another instance of the GA where jerk compensation was excluded from the fitness function. This allowed the effect of the jerk reduction to be quantified against a reference path, given a constant environment. It was found that the algorithm reduced the path's maximum jerk by an average of 192%. Ultimately, the actual jerk of the robotic manipulator will be a function of the robot's speed, however by reducing path jerk, the robot can navigate its trajectory at a higher speed without exceeding allowable jerk.

## VI. CONCLUSIONS

The results of path planning using genetic algorithms presented in this report show that genetic algorithms can effectively plan a path through a series of obstacles, and present an opportunity to improve on existing path planning techniques.

By selecting the appropriate fitness criteria, it is possible to bias a genetic search algorithm to find a path which satisfies multiple geometric conditions. In this implementation, the path was optimized to have a short path length, no collisions and minimal jerk.

The algorithm is on average able to produce a path that is roughly the same length (100.7%) as the shortest path as determined by the wavefront search method, while improving computation times by a factor of 100. 79% of the time the algorithm finds a path that is shorter than that produced by the wavefront method, with much longer paths occurring occasionally (in the range of 0.1% to 0.5% of the time). Collision avoidance is of critical importance in the application of path planning to industrial manipulators. This implementation of genetic algorithms could produce a collision free path 94% of the time. Finally, it was shown that the algorithm's jerk compensation was able to reduce path jerk by 11%, when compared to the same algorithm without jerk compensation.

Through a transformation from Cartesian space to the manipulator's joint configuration space, a method has been suggested to apply these path planning techniques to obstacle avoidance for industrial manipulators.

A main limitation of this algorithm is that since it makes use of ordered pair polynomial functions, there is a limited geography through which it is possible to evolve (a single 'y' value per 'x' value). Invalid solutions would therefore be found by the algorithm in a situation where it necessary for the path to reverse upon itself to avoid an obstacle. This is a fundamental limitation of a polynomial. In order to overcome this limitation, a function family would need to be selected that could double back across the same x-axis location.

Future study should be considered to determine the optimal function order for varying degrees of environment complexity. Increasing function order will allow for a more complex path, but at the cost of increased complexity and computational requirements. Further expansion of this technique into three degree-of-freedom and greater manipulators will greatly improve upon the industrial applicability of this technique. Validation of realizable improvements to path dynamics is also necessary to validate the effectiveness of jerk and other dynamic compensation techniques.

## VII. Appexdix - Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       MATLAB CODE - main.m         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Purpose: Executes genetic algorim to
    determine path across pre-defined
    envrionment. Used to collect datasets.

% Initialize file variables
simName = 'longtest';
simdir = strcat('sim_', simName);
mkdir(simdir);
fileName = strcat('gaData', int2str(
    runNumber),'_',datestr(now,'dd.mm.yyyy
    -HH.MM.SS'));

%% Initialize simulation params
numCspaces = 3;
numPointSets = 5;
colors = ['b' 'r' 'g' 'k' 'c'];
numSimulations = 500; %per points per
    cSpace
outputData = zeros(numSimulations, 17);

%% GA Configuration Params
PopulationSize = 75;
Generations = 50;

coefRangeMin = -500;
coefRangeMax = 500;

% GA Penalties (penalty weight relative)
obstacleWeight = 2;
lengthWeightFactor = 0.01;
lineResolution = 1;
jerkWeight = 1*0;

% GA Termination conditions
TerminationConvergenceTolerance = 0.001;
NumGensAvg = 10;

% GA Crossover parameters
crossoverFraction = 0.80;
eliteCount = 1;
crossoverFunction = @crossoverheuristic;
mutationFunction = @mutationadaptfeasible
    ;
fitnessScalingFunction = @fitscalingprop;

% GA Crossover selection criteria
tournamentSize = 2;
selectionFunction = {@selectiontournament
    , tournamentSize};

% Setup Plot
cSpaceFilenames = [ 'cSpace2',
                    'cSpace3',
                    'cSpace4',
                    'obsGrid'];

pointSetVector = [   7, 104; 349, 112;
                    15, 142; 238, 247;
                     7, 263; 311, 159;
                    91, 222; 314,  24;
                    58, 219; 184, 341;
                   155, 318; 352, 146;
                   116, 194; 206, 131;
                   156, 259; 286,  83;
```

```
                    48, 257; 208, 112;
                   151, 248; 308, 296;
                   105,  91; 319, 107;
                   143, 328; 264, 41;
                   120, 135; 263, 3;
                   65 , 208; 359, 314;
                   115, 229; 336, 47;]

% Iterate through each environment
for cSpaceIteration = 1:numCspaces

    cSpaceID = cSpaceIteration;
    fpath = strcat('../configSpace/',
        cSpaceFilenames(cSpaceIteration,:)
        , '.mat')
    obsGrid = importdata(fpath);

% Plot environment
    mapPlot = figure;
    hold on;
    [xDim yDim] = size(obsGrid);
    axis([0 xDim 0 yDim]);
    image(100*(1-obsGrid)');
    colormap(gray);

% Iterate through each start/end point
    configuration
    for pointsIteration = 1:numPointSets
    pointSetID = pointsIteration;
    cSpace = obsGrid;
    startPt = [pointSetVector(
        pointsIteration*2-1 + (
        cSpaceIteration -1)*10, 1),
        pointSetVector(pointsIteration*2-1
         + (cSpaceIteration -1)*10, 2)];
    endPt =   [pointSetVector(
        pointsIteration*2 + (
        cSpaceIteration -1)*10, 1),
        pointSetVector(pointsIteration*2 +
         (cSpaceIteration -1)*10, 2)];

% Plot start/end points
    figure(mapPlot);
    plot(startPt(1), startPt(2), 'g*', '
        MarkerSize',6);
    plot(endPt(1), endPt(2), 'g*', '
        MarkerSize',6);

    t = startPt(1):0.1:endPt(1);

% Repeat algorithm execution for dataset
        for j=1:numSimulations

% Number of variables in chromosome
        nvars = 4;

% Coefficient (Gene) cosntraints
        low = zeros(nvars,1);
        upp = zeros(nvars,1);
        range = zeros(2,nvars);

        mmin = coefRangeMin;
        mmax = coefRangeMax;

        for(i = 1:nvars)
            low(i) = mmin;
            upp(i) = mmax;
            range(1,i) = mmin;
            range(2,i) = mmax;
        end
        PopulationInitializationRange =
            range;

% Linear Equalities
        x1 = startPt(1);
        x2 = endPt(1);
        y1 = startPt(2);
        y2 = endPt(2);

        A_linEq = zeros(2, nvars);
        for(i = 1:nvars)
            A_linEq(1,i) = x1^(i-1);
            A_linEq(2,i) = x2^(i-1);
        end

        b_linEq = [y1; y2];

% Define GA options
        options = gaoptimset('
            PopInitRange',range);
        options = gaoptimset(options,'
            PopulationSize',PopulationSize
            );
        options = gaoptimset(options,'
            PopInitRange',
            PopulationInitializationRange)
            ;
        options = gaoptimset(options,'
            Generations',Generations);
        options = gaoptimset(options,'
            TolFun',
            TerminationConvergenceTolerance
            );
        options = gaoptimset(options,'
            StallGenLimit',NumGensAvg);
        options = gaoptimset(options,'
            SelectionFcn',
            selectionFunction);
        options = gaoptimset(options,'
            MutationFcn', mutationFunction
            );
        options = gaoptimset(options,'
            CrossoverFraction',
            crossoverFraction);
        options = gaoptimset(options,'
            CrossoverFcn',
            crossoverFunction);
        options = gaoptimset(options,'
            EliteCount', eliteCount);
        options = gaoptimset(options,'
            FitnessScalingFcn',
            fitnessScalingFunction);

% Execute GA
        tic;
        [x, Fval, exitFlag, Output] = ga(
            @(x) AKfitness(x,startPt,
            endPt, obstacleWeight,
            lengthWeightFactor, jerkWeight
            , lineResolution, j),nvars
            ,[],[],A_linEq,b_linEq,low,upp
            ,[],[],options);
        gaLengthTime = toc;

% Print GA results
        fprintf('Fitness␣Value␣=␣%g\n',
            Fval);
        fprintf('Generations␣␣␣=␣%g\n',
            Output.generations);
        format short;

        A = x(1);
        B = x(2);
        C = x(3);
        D = x(4);

        collision = 1;
        offScreen = 0;
        numCollisions = 0;
        djerk = 0;
        ddist = 0;
        dcoll = 0;

% Evaluate path
        y2 = @(t) sqrt(1 + (B + 2*C*t +
            3*D*t.^2).^2);
        ddist = integral(y2, startPt(1),
            endPt(1)) * lengthWeightFactor
            ;
        solutionLength = ddist;
        for i=startPt(1):lineResolution:
            endPt(1)
            y = A + B*i + C*i^2 + D*i^3;
            if (y > yDim || y < 0)
                numCollisions = 10000;
                dcoll = 10000;
                break;
            elseif (obsGrid(i,ceil(y)) ==
                1) %if within obstacle
                numCollisions =
                    numCollisions +
                    collision;
                dcoll = dcoll +
                    obstacleWeight * ceil(
                    abs(y_v));
            end
            y_v =   B   + 2*C*i + 3*D*i
                ^2;
            y_jerk =                 6*D
                ;
            if (y_jerk > djerk)
                djerk = y_jerk;
            end
        end

        maxJerk = djerk;

% Plot trajectory
        figure(mapPlot);
        hold on;
        plot(t,x(1)+x(2)*t + x(3)*t.^2 +
            x(4)*t.^3, colors(
            pointsIteration));

% Save GA/path results
        numGenerations = Output.
            generations;
        fitnessValue = Fval;
        outputData(j,:) = [cSpaceID
            pointSetID x solutionLength
            dcoll maxJerk numGenerations
            fitnessValue gaLengthTime
            PopulationSize startPt endPt];
        end

% Save results file
    save(strcat(simdir,'/', fileName, '.
        txt'), 'outputData', '-ASCII', '-
        append');

    end

% Save path figure
    figname = strcat(simdir, '/' ,
        cSpaceFilenames(cSpaceIteration,:)
        , '.fig');
    saveas(figure(cSpaceIteration),
        figname);
end

strcat('_Done',int2str(runNumber),'\n')

end
```

## VIII. Appexdix - Collected Data Sets

The experiement was run using the following datasets. Three configuration spaces were used which were generated from simulated robot environments, and on these environments five random sets of start and end points were generated. The three configuration spaces are presented, along with their respective points.

./figures/cspace2-eps-converted-to.pdf

Fig. 4. Configuration space #1, the points in table II were used with this space.

TABLE II
Point set #1

| X Value | Y Value |
|---|---|
| 7 | 104 |
| 349 | 112 |
| 15 | 142 |
| 238 | 247 |
| 7 | 263 |

TABLE III
Point set #2

| X Value | Y Value |
|---|---|
| 311 | 159 |
| 91 | 222 |
| 314 | 24 |
| 58 | 219 |
| 184 | 341 |

./figures/cspace3-eps-converted-to.pdf

Fig. 5. Configuration space #2, the points in table III were used with this space.

TABLE IV
Point set #3

| X Value | Y Value |
|---|---|
| 155 | 318 |
| 352 | 146 |
| 116 | 194 |
| 206 | 131 |
| 156 | 259 |

./figures/cspace4-eps-converted-to.pdf

Fig. 6. Configuration space #3, the points in table IV were used with this space.